TiGER: Tiny bandwidth key encapsulation mechanism for easy miGration based on RLWE(R)

Seunghwan Park, Chi-Gon Jung, Aesun Park,
Honggoo Kang, Joongeun Choi, and Janghyun Lee

Defense Counter-intelligence Command
horriblepaper@gmail.com, wjdclrhs@gmail.com,
aesunpark18@gmail.com,honggoonin@gmail.com,
joongeuntom@gmail.com, jhlee@gmail.com

Abstract. Designing a quantum-resistant (QR) key encapsulation mechanism (KEM) aims to replace KEM in legacy security protocols. Specifically, According to CNSA2.0, only the AES256 security level is allowed for national security purposes. Therefore, QR KEM should be designed to accommodate this security level in the legacy protocol. If QR KEM had a smaller bandwidth (ciphertext and public key size) than ECDH or DH, migrating would be easier. However, this seems impossible to Lattice-based KEM. Therefore, it makes sense to determine the threshold by analyzing the maximum bandwidth legacy protocols can acclimate. We specified the bandwidth threshold at 1,442 bytes based on IKEv2 (RFC7296), which has strict constraints on payload size during the initial exchange for key sharing. Unfortunately, Kyber, Saber, and Smaug do not meet this threshold at the AES256 security level.
In this paper, we propose TiGER, an IND-CCA secure QR KEM, which satisfies the threshold even at the AES256 security level. TiGER is the novel scheme to use RLWR for key generation and RLWE for encapsulation and decapsulation. Also, It combines various optimization techniques, such as sparse secret, error correction code, and byte modulus, from QR KEM schemes in the NIST competition. As a result, TiGER has a ciphertext size of 1,408 bytes and a public key size of 928 bytes at the AES256 security level, meeting the specified threshold. Furthermore, TiGER has shown a performance of 2.34-2.88 times better than Kyber.

Keywords: Key Encapsulation Mechanism · post-quantum cryptography · Migration to PQC · PQC for national security.

1 Introduction

QR KEM should easily apply to legacy security protocols (*e.g.*, TLS, IPsec). The best way to migrate is to design a QR KEM with similar performance, key and ciphertext size to legacy KEM (*e.g.*, ECDH, DH).

When we study KEMs based on LWE and its variants (Ring/Module LWE(R)), which were published in the NIST competition and KpqC (Korea post-quantum cryptography Competition), it appears impossible to design a QR KEM with a public key and ciphertext size smaller than DH or ECDH. Therefore, we should set a threshold for the suitable public key and ciphertext size that the current network environment and legacy security protocols can accept. We should then design a QR KEM smaller than that threshold.

In general, when using the IKEv2 protocol (RFC 7296) in an Ethernet with an MTU (Maximum Transmission Unit) of 1,500 bytes, the actual payload of 1,442 bytes can be sent except for the IPv4 header 20 bytes, UDP header 8 bytes, and IKE header 30 bytes (IKE header 28 bytes, generic payload header 1 byte, key exchange payload header 1 byte). If using IPv6, reduce it to 1,422 bytes. The first message, IKE-SA-INIT, cannot be fragmented (RFC 7383).

Taking into account 1,442 bytes as the threshold for the public key size and ciphertext size that legacy security protocols can accommodate, Kyber [8] with the AES192 security level recommended by FIPS 203 has no problem with QR migration. However, national security is an exception to this. For national security, QR KEM must use the AES256 security level specified in CNSA2.0 (Commercial National Security Algorithm Suite 2.0). Kyber has a public key and a ciphertext size of 1,568 bytes each at the AES256 security level. Saber [17], a NIST competition finalist, also has a ciphertext size of 1,472 bytes, and Smaug [12], which was submitted to the KpqC, also has the same problem with its public key and ciphertext sizes of 1,792 bytes and 1,472 bytes, respectively. Sable [34], an optimized version of Saber, has a ciphertext and public key size smaller than 1,442 bytes, but falls slightly below the AES256 security level. Therefore, when most of the QR KEMs published so far are used at the security level for national security, changes to existing legacy security protocols are required, such as using a complex and slow Hash and download technique [44] or switching to the IKE-INTERMEDIATE protocol [20,40]. It hinders QR migration and can lead to unexpected security issues.

1.1 Our Result

We propose TiGER a tiny bandwidth IND-CCA secure key encapsulation mechanism for easy migration based on RLWR and RLWE where the public key and the ciphertext size do not exceed 1,442 bytes on the AES256 security level. TiGER generates a key based on RLWR, encapsulation, and decapsulation based on RLWE, and compresses it to achieve a small bandwidth. To our knowledge, this is the first scheme. Intuitively, the rounding variant (Ring/Module-LWR) seems advantageous to achieve the smallest bandwidth, but since the error generation by rounding is done in the power of 2, it is hard to adjust the standard deviation of the error. Furthermore, since the technique for compressing ciphertext has been generalized to bring about a rounding effect, there is no need to use rounding as a default for encapsulation and decapsulation.

Also, we choose the RLWE(R) modulus $q = 256$ to cut-off a bandwidth. Since the hardness of RLWE(R) is determined by the dimension n and the σ/q, where σ is the standard deviation of noise distribution, choosing a small q is not a harmful hardness if determining a proper noise distribution [36,29]. If an

Table 1. Criteria evaluation for each parameter of TiGER.

Scheme	**Sizes** (bytes)			**Security**			**Cycle** (CPU cycle counts) [3]		
	sk	**pk**	**ct**	**Level**	**Sec.**[1]	**DFP**[2]	**Keygen**	**Encap**	**Decap**
TiGER128	134	480	640	AES128	127	2^{-132}	55K	51K	47K
TiGER192	178	928	1,280	AES192	221	2^{-187}	66K	82K	76K
TiGER256	263	928	1,408	AES256	257	2^{-163}	73K	105K	103K

[1] given in classical core-SVP.
[2] Decryption Failure Probability
[3] AMD Ryzen7 5700G CPU, Ubuntu 22.04.1, GCC 11.3.0 with option -O3

error is added to a small q and even ciphertext compression is performed, the decryption failure rate will be very high. We solve this problem with error correction codes `XEf` [9] and `D2` [6]. In particular, the unique characteristics of the ring $f(X)=X^n+1$ in $R_q := \mathbb{Z}_q[X]/(f(X))$ are a good combination with the error correction code. That is, in the X^n+1, n is constrained to the power of 2, and since it is sufficiently more significant than the required message size (128, 192, 256 bits), this buffer is used as a valuable space for redundancy bits. We use a sparse ternary uniform distribution with hamming weights for the secret/error distribution. It chose to enjoy being able to fine-tune the standard deviation through hamming weights while minimizing the propagation of errors. And allows the replacement of polynomial multiplication with bit-wise operations.

As a result, even at the AES256 security level, TiGER maintains a public key and ciphertext size under 1,442 bytes, making it ideal for national security applications with legacy security protocols. The secret key size is tiny, making it suitable for resource-constrained devices. Also, its performance is excellent. Table 1 summarizes the evaluation of TiGER using evaluation criteria.

Advantages

- **Compact :** TiGER has a tiny ciphertext and public key size compared to the KpqC candidates and the finalists of the NIST competition based on LWE (and variants). The bandwidth (size of public key + ciphertext) of TiGER128 with AES128 security level is 1,120 bytes, which is 40% smaller than Kyber, and 20% smaller than Smaug. In AES256 security level for national security, TiGER256 is 34% smaller than Kyber and 39% smaller than Smaug.
- **Easily migration :** Our goal in designing QR KEM is to replace key exchange algorithms in legacy security protocols. We claim that migration to QR KEM may be complex if the public key or ciphertext size is more significant than 1,442 bytes (IKEv2 over Ethernet). TiGER256 with AES256 security level has a ciphertext and public key size of 1,408 bytes and 928 bytes, smaller than 1,442 bytes, and a free space has 34 bytes. This free space can be used for additional header options in real-world network configurations. In addition, TiGER has a tiny secret key (134, 178, and 263 Bytes

for each security level, respectively). Therefore, it is advantageous when secret keys must be protected in resource-constrained devices or Hardware security modules (HSM).

- **High performance :** TiGER samples the secret **s** from a sparse ternary uniform distribution, making polynomial multiplication efficient. TiGER is 2.34-2.88 times faster than Kyber and 1.26-1.71 times faster than Smaug.
- **Friendly for SIMD :** TiGER is friendly for SIMD, such as AVX2 and NEON, due to the modulus q=256 (8 bits). For example, C intrinsic data types `__m256i` can be stored in a 32-dimensional 8-bit integer so that only 16 `__m256i` data types can express TiGER parameters.

Limitations

- The side-channel analysis attack surface is larger than None-ECC. However, our choice of `XEf` is at least resistant to timing attacks. Some attacks assume a more powerful attacker, but we believe that is not a specific problem to ours and will be resolved wisdom of crowds (as is the case of RSA and ECC).

Table 2. Compared to other schemes.

Schemes	Size (bytes)		Security	DFP	Performance (cycle ratio)			
	sk	bandwidth	(core-SVP)	(*log*)	Total	Key.	Enc.	Dec.
NIST Security Level 1 (AES128)								
TiGER128	**134**	**1,120**	**127**	**-132**	**1**	**1**	**1**	**1**
Kyber512	1,632	1,568	118	-139	2.34	1.65	2.34	3.13
LightSaber	832	1,408	118	-120	1.30	1.12	1.34	1.49
LightSable	800	1,280	114	-139	1.22	1.01	1.23	1.45
Smaug128	176	1,344	120	-120	1.26	1.44	1.23	1.09
NIST Security Level 3 (AES192)								
TiGER192	**178**	**2,208**	**221**	**-187**	**1**	**1**	**1**	**1**
Kyber768	2400	2,272	183	-164	2.51	2.34	2.25	2.92
Saber	1248	2,080	189	-136	1.57	1.59	1.47	1.66
Sable	1152	1,920	185	-143	1.46	1.46	1.35	1.58
Smaug192	236	2,112	181	-136	1.30	1.74	1.11	1.11
NIST Security Level 5 (AES256)								
TiGER256	**263**	**2,336**	**257**	**-163**	**1**	**1**	**1**	**1**
Kyber1024	3,168	3,136	256	-174	2.88	3.23	2.51	3.02
FireSaber	1,664	2,784	260	-165	1.94	2.26	1.77	1.90
FireSable	1,632	2,688	223	-208	1.82	2.10	1.66	1.79
Smaug256	218	3,264	264	-167	1.71	2.49	1.45	1.43

2 Preliminaries

2.1 Public Key Encryption

A public key encryption (PKE) scheme is a cryptographic system that uses a pair of a public key and a corresponding private key. The security of PKE depends on maintaining the confidentiality of the private key. The public key can be publicly distributed, and anyone can encrypt using the public key that only the user who has the private key can decrypt it. The following is the syntax of PKE.

Definition 1 (PKE). *A PKE scheme consist of three algorithms **KeyGen**, **Encryption**, **Decryption** which are defined as follows:*

***KeyGen**(1^λ): The key generation algorithm takes as input a security parameter 1^λ. It outputs a public key **pk** and a private key **sk**.*
***Encryption(pk, M)**: The encryption algorithm takes as input the public key **pk** and a message $\boldsymbol{M} \in \mathcal{M}$. It outputs a ciphertext **c**.*
***Decryption(sk, c)**: The decryption algorithm takes as input the private key **sk** and the ciphertext **c**. It outputs the message **M** or $\perp$.*

*The correctness property of PKE is defined as follows: For all **pk** and **sk** generated by **KeyGen**(1^λ), **c** generated by **Encryption(pk, M)** for **M**, it is required that*

- *If **sk** is valid, then **Decryption(sk, c)** = **M**.*
- *If **sk** is invalid, then **Decryption(sk, c)** = $\perp$.*

2.2 Key Encapsulation Mechanism

A key encapsulation mechanism (KEM) is used to share the secret key of symmetric key encryption systems. The sender generates a ciphertext for sharing the secret key and sends it to the receiver. The receiver decapsulates the ciphertext and generates a secret key to be used for symmetric key encryption. The following is the syntax of KEM.

Definition 2 (KEM). *A KEM scheme consist of three algorithms **KeyGen**, **Encapsulation**, **Decapsulation** which are defined as follows:*

***KeyGen**(1^λ): The key generation algorithm takes as input a security parameter 1^λ. It outputs a public key **pk** and a private key **sk**.*
***Encapsulation(pk)**: The encapsulation algorithm takes as input the public key **pk**. It outputs a ciphertext **c** and a shared key **K**.*
***Decapsulation(sk, c)**: The decapsulaiton algorithm takes as input the private key **sk** and the ciphertext **c**. It outputs the shared key **K** or $\perp$.*

*The correctness property of KEM is defined as follows: For all **pk** and **sk** generated by **KeyGen**(1^λ), **c** and **K** generated by **Encapsulation(pk)**, it is required that*

- *If **sk** is valid, then **Decapsulation(sk, c)** = $\hat{\boldsymbol{K}}$ such that $\boldsymbol{K} = \hat{\boldsymbol{K}}$.*
- *If **sk** is invalid, then **Decapsulation(sk, c)** = $\hat{\boldsymbol{K}}$ or $\perp$ such that $\boldsymbol{K} \neq \hat{\boldsymbol{K}}$.*

2.3 Related Works

Lattice-based Public-Key Encryption. Due to the threat of quantum computers to the classical cryptography such as RSA or Diffie-Hellman, lattice-based cryptography is attracting attention as one of various post-quantum cryptography. The first lattice-based cryptographic construction was introduced by M. Ajtai [1]. And, the first lattice-based public key encryption (PKE) that called NTRU scheme is proposed by J. Hoffstein et al. [21]. The security of thier PKE scheme was not proven under worst-case hardness assumptions. In 2005, O. Regev [38] introduced the first lattice-based PKE scheme with the Learning With Errors (LWE) problem. The LWE problem is as difficult to solve as the worst-case lattice problems. Since then, a variety of lattice problems such as Learning With Rounding (LWR), Ring-LWE (RLWE), Ring-LWR (RLWR), etc. have been proposed [30,37,31,10], and these problems have been widely used to design public key cryptography. Also, after the process of standardizing post-quantum cryptography (PQC), many other lattice-based PKE schemes such as Kyber [8], NewHope [6], LAC [29], Round5 [9], Saber [17], Sable [34] and Smaug [12] assuming the hardness of lattice problems have been proposed.

CCA-secure KEM. Chosen-ciphertext attack (CCA) is a security model that allows an adversary to obtain the plaintext corresponding to the chosen ciphertext. Because the process of establishing a session key in communications is similar to accessing a decryption oracle by an adversary. It is very important to design the key encapsulation mechanism (KEM) considering the chosen-cipertext attack. Fujisaki and Okamoto [19] introduced a generic transformation method that can drive the chosen-cipertext secure KEM scheme from the chosen-plaintext secure PKE scheme. And, this method has been widely used to design many cryptographic algorithms. Recently, the Fujisaki-Okamoto transformation method in consideration of a quantum computing environment has been proposed [42,23,22]. Jiang et al. [23] presented Fujisaki-Okamoto transformation method that was proven tight security reductions in the quantum random oracle model.

3 Design rationale

TiGER is an IND-CCA-secure Key Encapsulation Mechanism (KEM) based on the hardness of solving the learning-with-rounding and learning-with-errors problem over Ring lattices (RLWR and RLWE problem). The TiGER scheme is constructed in two-steps: we first introduce and IND-CPA-secure PKE (Public Key Encryption) scheme encrypting messages of a length of 16 or 32 bytes. We then use a Fujisaki–Okamoto (FO) transform by Jiang et al. [23] to construct the IND-CCA-secure KEM.

3.1 Ring-LWR and Ring-LWE

RLWE(R) are structured lattices, but specific attacks that apply only to these are unknown and have been well-studied for a relatively long time. Furthermore, since it is used in homomorphic encryption with sparse secret distribution [13], which has recently been in the spotlight in protecting personal information and private AI, it will certainly be a structure that will be actively studied in the future. We expect that this heritage and research trends will exert collective intelligence in security analysis, and judged it to be an appropriate choice for KEM for national security.

whereas NIST competition finalists are based on Module-Lattice. The biggest advantage of Module-lattice can be fine-tuned according to the security level by adjusting the module dimension($k = 2, 3, 4, 5$) while fixing the ring dimension small ($n = 256$). However, this advantage is only valid for the AES192 security level. For Kyber, Saber, and Smaug, $n \times k = 768$ is the AES192 security level, and at the AES256 security level, they are all set to $n \times k = 1024$ or higher. Therefore, the meaning is lost in AES256, which is the security level for national security purposes that we aim for.

Choice of the Ring We use $f(X)=X^n + 1$ in $R_q := \mathbb{Z}_q[X]/(f(X))$, where n is the power of 2. It is the common choice used by most of the Ring-lattice submitted to NIST competitions. The common ring has the advantage in that the polynomial modular reduction operation is straightforward, and there have been no known attacks exploit it [27]. On the other side of this choice, $f(X)=X^{n+1}+1$ in [9], where $n+1$ is prime and $f(X)=X^n-X^{n/2}+1$ in [35], where $n = 2^a3^b$(a and b are positive integer) have the flexibility to select the appropriate degree n for each security level. Although we observed these recent studies meaningfully, we decided to enjoy the aspect of conservative security that there was no particular weakness even though $f(X)=X^n+1$ was well studied and the attribute of making the probability of decoding error the smallest. In addition, the constraint of degree n is tricky to witness as a disadvantage because it provides a valuable buffer space in terms of employing the error correction code.

Choice of modulus All integer modulus in the scheme are power of 2. It improve performance by replacing $\lfloor (p/q) \cdot \mathbf{x} \rceil$ with ADD and AND operations [14]. Especially, fixed $q = 256$ is a byte size. The first proposed the choice of modulus q below the byte size with $q = 251$ in LAC [29], which won the PQC competition in China. In addition, LizarMong [24] designed the scheme with $q = 256$, which is the power of 2 like ours, and then LAC also added $q = 256$ parameter as an option during the NIST competition. Intuitively, this choice enjoys a small bandwidth and improved performance. It also provides efficient modulo operation and memory usage and is suitable for single instruction multiple data (SIMD) implementations such as AVX2 and NEON. Even though the modulus is small, it can not affect the security since we maintain the error rate by selecting proper error distribution [29,36]. The modulus p used for RLWR and the modulus k_1, k_2 used for ciphertext compression are also the power of two.

3.2 Adopt error correction code

$R_q := Z_q[X]/(X^n + 1)$ constrains n to the power of 2, so choose $n = 512$ or $n = 1024$ depending on the security level. Since general RLWE(R) schemes map one bit message to one coefficient in R_q, n is larger than the required size of the message δ (128, 192, 256 bits). We claim that RLWE(R) has the best compatibility with error correction code (ECC) regarding message processing. A redundancy bit is inevitably required for error correction. n larger than the length of the message δ can be used as a buffer sufficient to use the redundancy bit. In particular, Because the RLWE(R) is a variant to maximize performance and reduce the size (ciphertext, public key) of the LWE, RLWE(R) with comparable performance and size to less structured MLWE is meaningless [4]. Thus, a small q selection is needed in RLWE(R), increasing the decoding failure probability. So, the combination of RLWE(R) and ECC is beautiful because ECC can be an excellent choice to overcome these drawbacks. We use the well-studied `XEf` [9] and `D2` [6] to utilize the buffer space. Since `XEf` avoids table look-up and branch conditions, it resists timing attacks [9]. That is, a 256-bit message is encoded with `XE5` to make a 512bit code word (234 redundancy bits and 22 padding bits), and `D2` encodes the code word to make a 1,024bit $\hat{\mathbf{M}}$. Decoding is in reverse order.

Remark In LWE schemes that include errors, error correction codes are a very reasonable approach. It has been proven in several studies that error correction codes have an apparent effect in reducing the bandwidth and decrypt failure probability of LWE-based schemes [18,39,26,28,24]. On the other hand, many reports have shown that it is vulnerable to side-channel analysis [15]. This was highlighted because LAC, the winner of China's PQC competition, used BCH rather than a constant time implementation. However, it must be remembered that finally implemented constant-time BCH [43]. The `XEf` we chose is an ECC made suitable for the LWE scheme by eliminating concerns about side-channel analysis in advance [39]. A Korean proverb says, "Maggots cannot be an excuse for not making soybean paste." Although we know that error correction codes are handy in LWE systems, we do not think rejecting them without improving their use is reasonable.

3.3 Sparse ternary uniform distribution with hamming weight

We use a sparse ternary secret with a hamming weight [14] and [9] proved the hardness of the sparse ternary secret variants RLWE(R). Multiplication of sparse ternary secret polynomials can be replaced with bit operation to improve performance [2,27]. It also maintains correctness by preventing decryption errors from increasing and can select the optimized standard deviation for each security level by finely adjusting the hamming weight and has the advantage of having resistance to high hamming weight attacks [29] that manipulate the centered binomial distribution.

4 Specification

4.1 Notation

Let $\mathbb{Z}$ be the ring of rational integers. We define for an $x \in \mathbb{R}$ the rounding function $\lfloor x \rceil$, where $\lfloor x \rceil$ means the nearest integer to the x. We denote $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ which means the ring of integer polynomial modulo $(X^n + 1)$, where each coefficient is reduced modulo q. n is a positive integer expressed as a power of two, which means the dimension of RLWE samples. q and p mean modulus for RLWE and RLWR, respectively. We also use the modulus k_i, where $i = 1, 2$, for ciphertext compression. We use p, q, k_1, and k_2 which are all the power of 2. Bold lower-case letters represent polynomials with coefficients in R_q. Multiplication in R_q is represented by $*$. $\lfloor \mathbf{a} \rceil$ is the rounding to the nearest integer for each coefficient in the polynomial $\mathbf{a}$. $x \parallel y$ is the concatenation of x and y. $HWT_n\,(h, Seed)$ is the uniform distribution over the subset of $\{-1, 0, 1\}^n$ whose elements contain $n - h$ number of zeros, and is generated using $Seed$. `SHAKE256`(m, len) is a hash function that receives m and outputs a byte-string of the length len. eccENC and eccDEC are functions for encoding and decoding using the error correction codes.

4.2 Specification of TiGER.CPAPKE

Algorithm 1 `IND-CPA.KeyGen`

```
Input: The set of public $parameters$
Output: Public key $pk = (Seed_a \parallel \mathbf{b})$, Private Key $sk = (\mathbf{s})$
1: $Seed_a \xleftarrow{\$} \{0,1\}^{256}$
2: $Seed_s \xleftarrow{\$} \{0,1\}^{256}$
3: $\mathbf{a} \leftarrow \texttt{SHAKE256}(Seed_a, n/8)$
4: $\mathbf{s} \leftarrow HWT_n(h_s, Seed_s)$
5: $\mathbf{b} \leftarrow \lfloor (p/q) \cdot \mathbf{a} * \mathbf{s} \rceil$
6: $pk \leftarrow (Seed_a \parallel \mathbf{b})$ and $sk \leftarrow \mathbf{s}$
7: return $pk, sk$
```

Algorithm 2 `IND-CPA.Encryption`

```
Input: $pk$, Message $\mathbf{M} \in \{0,1\}^d$ ; Coin $w \xleftarrow{\$} \{0,1\}^{256}$
Output: Ciphertext $\mathbf{c} = (\mathbf{c_1} \parallel \mathbf{c_2})$
1: $\mathbf{r} \leftarrow HWT_n(h_r, w)$
2: $Seed_{e1} \leftarrow (w + Nonce)$
3: $Seed_{e2} \leftarrow (w + Nonce + 1)$
4: $\mathbf{e_1} \leftarrow HWT_n(h_{e1}, Seed_{e1})$ and $\mathbf{e_2} \leftarrow HWT_n(h_{e2}, Seed_{e2})$
5: $Seed_a, \mathbf{b} \leftarrow$ Parsing$(pk)$
6: $\mathbf{a} \leftarrow \texttt{SHAKE256}(Seed_a, n/8)$
7: $\mathbf{c_1} \leftarrow \lfloor (k_1/q) \cdot (\mathbf{a} * \mathbf{r} + \mathbf{e_1}) \rceil$
8: $\mathbf{c_2} \leftarrow \lfloor (k_2/q) \cdot ((q/2) \cdot$ eccENC$(\mathbf{M})$ + $((q/p) \cdot \mathbf{b}) * \mathbf{r} + \mathbf{e_2}) \rceil$
9: $\mathbf{c} \leftarrow (\mathbf{c_1} \parallel \mathbf{c_2})$
10: return $\mathbf{c}$
```

Algorithm 3 `IND-CPA.Decryption`

```
Input: $sk$, Ciphertext $\mathbf{c} = (\mathbf{c_1} \parallel \mathbf{c_2})$
Output: Message $\hat{\mathbf{M}}$
1: $\mathbf{c_1}, \mathbf{c_2} \leftarrow$ Parsing($\mathbf{c}$) and $\mathbf{s} \leftarrow sk$
2: $\hat{\mathbf{M}}' \leftarrow \lfloor (2/q) \cdot ((q/k_2) \cdot \mathbf{c_2} - ((q/k_1) \cdot \mathbf{c_1}) * \mathbf{s}) \rceil$
3: return $\hat{\mathbf{M}} \leftarrow$ eccDEC($\hat{\mathbf{M}}'$)
```

4.3 Specification of TiGER.CCAKEM

We design IND-CCA KEM using the transformation technique by Jiang et al. [23]. We use a hash function $H : \{0,1\}^* \to \{0,1\}^{256}$, and a hash function $G : \{0,1\}^* \to \{0,1\}^*$ for Jiang's transformation technique.

Algorithm 4 `IND-CCA-KEM.KeyGen`

```
Input: The set of public $parameters$
Output: Public Key $pk = (Seed_a \parallel \mathbf{b})$, Private Key $sk = (sk_{cpa} \parallel \mathbf{u})$
1: $pk, sk_{cpa}$ := IND-CPA.KeyGen($parameters$)
2: $\mathbf{u} \xleftarrow{\$} R_2$
3: return $pk, sk \leftarrow (sk_{cpa} \parallel \mathbf{u})$
```

Algorithm 5 `IND-CCA-KEM.Encapsulation`

```
Input: $pk$
Output: Ciphertext $\mathbf{c} = (\mathbf{c_1} \parallel \mathbf{c_2})$, Shared Key $\mathbf{K}$
1: $\delta \xleftarrow{\$} \{0,1\}^d$
2: $\mathbf{c}$ := IND-CPA.Encryption($pk$, $\delta$ ; $H(\delta, H(pk))$)
3: $\mathbf{K} \leftarrow G(H(\mathbf{c}), \delta)$
4: return $\mathbf{c}, \mathbf{K}$
```

Algorithm 6 `IND-CCA-KEM.Decapsulation`

```
Input: $pk$, $sk$, Ciphertext $\mathbf{c}$
Output: Shared Key $\mathbf{K}$
1: $\mathbf{s}, \mathbf{u} \leftarrow$ Parsing($sk$)
2: $\hat{\delta}$ := IND-CPA.Decryption($\mathbf{s}, \mathbf{c}$)
3: $\hat{\mathbf{c}}$ := IND-CPA.Encryption($pk$, $\hat{\delta}$ ; $H(\hat{\delta}, H(pk))$)
4: if $\mathbf{c} = \hat{\mathbf{c}}$ then $\mathbf{K} \leftarrow G(H(\mathbf{c}), \hat{\delta})$ else $\mathbf{K} \leftarrow G(H(\mathbf{c}), \mathbf{u})$
5: return $\mathbf{K}$
```

4.4 Parameter sets

We construct a TiGER128 that satisfies security level 1 (AES128), a TiGER192 that satisfies security level 3 (AES192) and TiGER256 that satisfies security level 5 (AES256) as required by the NIST competition. Table 3 shows the detailed parameters of each security level and the bandwidth according to each security level is summarized in Table 4.

Table 3. The detail parameters for each security level

parameters	n	q	p	k_1	k_2	h_s	h_r	h_{e1}	h_{e2}	d	f
TiGER128	512	256	128	128	8	104	104	32	32	128	3
TiGER192	1024	256	128	128	8	116	116	32	32	256	5
TiGER256	1024	256	128	256	8	184	184	256	32	256	5

Table 4. Size of pk, sk, and ciphertext (bytes)

parameters	**Ciphertext**	**Public key**	**Secret key**[*]
TiGER128	640	480	134
TiGER192	1,280	928	178
TiGER256	1,408	928	263

[*] sk_{cpa} can be encoded by storing only non-zero indexes. Thus, we adopted sk can be compressed with $encoding(sk_{cpa})$, a flag of 1 or -1, and $\mathbf{u}$ (for IND-CCA KEM).

n is the dimension of the lattice, q is the modulus of RLWE, p is the modulus of RLWR, k_1 and k_2 are the modulus used for ciphertext compression, h_s, h_r is the hamming weight of the secret key and the ephemeral secret used to encryption. h_{e_1} and h_{e_2} are the hamming weight of error distribution. d is the length of the message, which is related to the security level. f is the number of error bits fixed by error correction code.

4.5 Correctness

The following shows the correctness of our PKE scheme. Let $\hat{\mathbf{M}}$ be an encoded message from eccENC ($\mathbf{M}$), where $\mathbf{M} \in \mathcal{M}$. Let ψ be a positive integer parameter and $\mathbf{s}$, $\mathbf{r}$, $\mathbf{e_1}$, $\mathbf{e_2}$ are randomly chosen from a same distribution $\mathcal{D}$. The value of $\mathbf{e_b}$ is :

$$\mathbf{e_b} = ((\frac{q}{p}) \cdot \lfloor (\frac{p}{q}) \cdot \mathbf{b} \rceil) - \mathbf{b},$$

for some $\mathbf{e_b} \in R_\psi$. The value of $\mathbf{e_{c_1}}$ is :

$$\mathbf{e_{c_1}} = ((\frac{q}{k_1}) \cdot \lfloor (\frac{k_1}{q}) \cdot \mathbf{c_1} \rceil) - \mathbf{c_1},$$

for some $\mathbf{c_1} \in R_\psi$. The value of $\mathbf{e_{c_2}}$ is :

$$\mathbf{e_{c_2}} = ((\frac{q}{k_2}) \cdot \lfloor (\frac{k_2}{q}) \cdot \mathbf{c_2} \rceil) - \mathbf{c_2},$$

for some $\mathbf{c_2} \in R_\psi$. Then, the decryption process is as follows:

$$\begin{aligned}
&\lfloor(\frac{2}{q}) \cdot \big((\frac{q}{k_2}) \cdot \mathbf{c_2} - ((\frac{q}{k_1}) \cdot \mathbf{c_1}) * \mathbf{s}\big)\rceil \\
=&\lfloor(\frac{2}{q}) \cdot \big((\frac{q}{k_2}) \cdot \lfloor(\frac{k_2}{q}) \cdot ((\frac{q}{2}) \cdot \hat{\mathbf{M}} + ((\frac{q}{p}) \cdot \lfloor(\frac{p}{q}) \cdot \mathbf{a} * \mathbf{s}\rceil) * \mathbf{r} + \mathbf{e_2})\rceil \\
&- ((\frac{q}{k_1}) \cdot \lfloor(\frac{k_1}{q}) \cdot (\mathbf{a} * \mathbf{r} + \mathbf{e_1})\rceil) * \mathbf{s}\big)\rceil \\
=&\lfloor(\frac{2}{q}) \cdot \big(((\frac{q}{2}) \cdot \hat{\mathbf{M}} + \mathbf{e'_b r} + \mathbf{e'_2} + \mathbf{e_{c_2}}) - (\mathbf{e'_1 s} + \mathbf{e_{c_1} s})\big)\rceil \\
=&\hat{\mathbf{M}},
\end{aligned}$$

Let $\mathbf{f} = (\mathbf{e'_b r} + \mathbf{e'_2} + \mathbf{e_{c_2}}) - (\mathbf{e'_1 s} + \mathbf{e_{c_1} s})$, We have that the error rate is $\hat{\epsilon} = 1 - Pr[\lfloor -q/2\rceil < \mathbf{f_i} + \mathbf{f_j} < \lfloor q/2\rceil]$, since our PKE scheme uses `D2` encode that encodes from one message bit to two coefficients. Using the decryption failure probability estimator of M. Albrecht[1], we obtain that the error rate of each message bit is $2^{-42.06}$. Our PKE scheme uses `XE3` in security level 1 to correct 3-bit errors, we have that

$$\epsilon = 1 - \big(\sum_{f=0}^{3} \binom{512}{f} \cdot ((2^{-41.06})^f) \cdot (1 - 2^{-41.06})^{512-f}\big) \approx 2^{-132.86}.$$

Then, our PKE scheme is (1-ϵ)-correct with $\epsilon < 2^{-\lambda}$, where security parameter λ is 128. The following Table 5 shows the decryption failure probability of our PKE scheme's each parameters. Note that there is a hidden margin in our decryption failure probability calculation. `XE`f accurately corrects errors of f-bits and can correct errors more than f with a high probability. Experimentally `XE5` corrects 99.4% of random 6-bit errors and 97.0% of random 7-bit errors [39].

Table 5. Decryption Failure Probability

parameters	**Bit error rate**	**Decryption Failure Probability**	f^*
TiGER128	$2^{-41.06}$	$2^{-132.86}$	3
TiGER192	$2^{-39.60}$	$2^{-187.10}$	5
TiGER256	$2^{-35.62}$	$2^{-163.21}$	5

* Let f be the bit length to be corrected by using the `XE`f.

[1] https://bitbucket.org/malb/lwe-decryption-failure.git

5 Security

5.1 Security definition

The following is the formal definition of the IND-CPA security.

Definition 3 (IND-CPA). *Let PKE = (**KeyGen**, **Encrypt**, **Decrypt**) be a public-key encryption scheme. The security of PKE under chosen plaintext attacks is defined in terms of the following experiment between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:*

```
$\textbf{\textit{Exp}}^{IND\text{-}CPA}_{PKE,\mathcal{A}}(\lambda)$
    1. $(pk, sk) \leftarrow \textbf{\textit{KeyGen}}(1^\lambda)$;
    2. $(M_0, M_1) \leftarrow \mathcal{A}$;
    3. $\mathcal{C}$ flips a random coin $b \in \{0,1\}$;
    4. $ct \leftarrow \textbf{\textit{Encrypt}}(PK, M_b)$;
    5. $b' \leftarrow \mathcal{A}$;
        If $b = b'$, return 1.
        Otherwise, return 0.
```

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}^{IND\text{-}CPA}_{PKE,\mathcal{A}}(\lambda) = \left|\Pr[b = b'] - \frac{1}{2}\right|$ where the probability is taken over all the randomness of the experiment. A PKE scheme is secure in the security model under chosen plaintext attacks if for all adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above experiment is negligible in the security parameter λ.

Security Assumption We define decisional Ring Learning With Errors (RLWE) problem and decisional Ring Learning With Rounding (RLWR) problem. Let R_q, R_p denote the rings $\mathbb{Z}_q[x]/(g(x))$, $\mathbb{Z}_p[x]/(g(x))$, where $g(x)$ is an irreducible polynomial of degree n.

Definition 4 (decisional RLWE). *Let n, q be positive integers. Let R_q be polynomial ring constructed by $g(x)$, and let $\mathfrak{D}_s$ be a distribution over R_q. A decisional RLWE problem $\mathbf{RLWE}_{n,q}(\mathfrak{D}_s)$ is to distinguish uniformly random $(\mathbf{a}, \mathbf{u}) \in R_q \times R_q$ and $(\mathbf{a}, \mathbf{b} = \mathbf{a} * \mathbf{s} + \mathbf{e}) \in R_q \times R_q$, where $\mathbf{a}$ is uniform randomly chosen polynomial, $\mathbf{e}$ is chosen from error distribution, and $\mathbf{s}$ is a secret polynomial. Then, the advantage of an adversary $\mathcal{A}$ in solving the decisional RLWE problem $\mathbf{RLWE}_{n,q}(\mathfrak{D}_s)$ is defined as follows:*

$$\textbf{\textit{Adv}}^{RLWE}_{n,q}(\mathcal{A}) = |\mathsf{Pr}[\mathcal{A}(\mathbf{a}, \mathbf{b}) = 1] - \mathsf{Pr}[\mathcal{A}(\mathbf{a}, \mathbf{u}) = 1]|.$$

Definition 5 (decisional RLWR). *Let n, q, p be positive integers such that $q > p$. Let R_q, R_p be polynomial rings constructed by $g(x)$, and let $\mathfrak{D}_s$ be a distribution over R_q. A decisional RLWR problem $\mathbf{RLWR}_{n,q,p}(\mathfrak{D}_s)$ is to distinguish uniformly random $(\mathbf{a}, \mathbf{u}) \in R_q \times R_p$ and $(\mathbf{a}, \mathbf{b} = \lfloor (p/q) \cdot (\mathbf{a} * \mathbf{s}) \rceil) \in R_q \times R_p$, where $\mathbf{a}$ is uniform randomly chosen polynomial, and $\mathbf{s}$ is a secret polynomial. Then. the advantage of an adversary $\mathcal{A}$ in solving the decisional RLWR problem $\mathbf{RLWR}_{n,q,p}(\mathfrak{D}_s)$ is defined as follows:*

$$\boldsymbol{Adv}^{RLWR}_{n,q,p}(\mathcal{A}) = |\Pr[\mathcal{A}(\mathbf{a}, \mathbf{b}) = 1] - \Pr[\mathcal{A}(\mathbf{a}, \mathbf{u}) = 1]|.$$

5.2 Formal Security

Security of IND-CPA PKE We prove that our PKE scheme is IND-CPA secure under the RLWE assumption and the RLWR assumption.

Theorem 1 (IND-CPA PKE). *The above PKE scheme is secure under chosen plaintext attacks if the the RLWE assumption and the RLWR assumption holds. That is, for any PPT adversary $\mathcal{A}$, we have that $\boldsymbol{Adv}^{IND-CPA}_{PKE}(\mathcal{A}) \leq \boldsymbol{Adv}^{RLWE}_{n,q}(\mathcal{B}) + \boldsymbol{Adv}^{RLWR}_{n,q,p}(\mathcal{B})$.*

Proof. The security proof consists of the sequence of hybrid games: The first game will be the original security game and the last one will be a game such that the adversary has no advantage. Let $\hat{\mathbf{M}}$ = eccENC ($\mathbf{M}$). We define the games as follows:
Game G_0 **:** This game is the original security game. In this game, the public key and the ciphertext are properly generated. $\mathcal{D}_0$ is described as follows:

$$\begin{aligned}\mathcal{D}_0 = \{&pk = (Seed_a \parallel \mathbf{b} = \lfloor (\frac{p}{q}) \cdot \mathbf{a} * \mathbf{s} \rceil), \\ &\quad ct = (\lfloor (\frac{k_1}{q}) \cdot (\mathbf{a} * \mathbf{r} + \mathbf{e_1}) \rceil), \lfloor (\frac{k_2}{q}) \cdot ((\frac{q}{2}) \cdot \hat{\mathbf{M}}_b + ((\frac{q}{p}) \cdot \mathbf{b}) * \mathbf{r} + \mathbf{e_2}) \rceil \}.\end{aligned}$$

Game G_1 **:** In the next game, $\mathbf{b}$ in the public key is replaced with uniformly random polynomial in R_p.$\mathcal{D}_1$ is described as follows:

$$\begin{aligned}\mathcal{D}_1 = \{&pk = (Seed_a \parallel \mathbf{b} \overset{\$}{\leftarrow} R_p), \\ &\quad ct = (\lfloor (\frac{k_1}{q}) \cdot (\mathbf{a} * \mathbf{r} + \mathbf{e_1}) \rceil), \lfloor (\frac{k_2}{q}) \cdot ((\frac{q}{2}) \cdot \hat{\mathbf{M}}_b + ((\frac{q}{p}) \cdot \mathbf{b}) * \mathbf{r} + \mathbf{e_2}) \rceil \}.\end{aligned}$$

Therefore, $|\Pr[S_0]$ - $\Pr[S_1]| \leq \mathbf{Adv}^{RLWR}_{n,q,p}(\mathcal{B})$.
Game G_2 **:** In the final game, $\mathbf{a} * \mathbf{r} + \mathbf{e_1}$ and $(q/p) \cdot \mathbf{b}) * \mathbf{r} + \mathbf{e_2}$ are replaced with uniformly random polynomial in R_q. Let $\mathbf{u} \overset{\$}{\leftarrow} R_q$ and $\mathbf{v} \overset{\$}{\leftarrow} R_q$. $\mathcal{D}_2$ is described

as follows:

$$\begin{aligned}\mathcal{D}_2 = \{pk = (Seed_a \parallel \mathbf{b} \overset{\$}{\leftarrow} R_p),\\ ct = (\lfloor(\frac{k_1}{q}) \cdot \mathbf{u}\rceil), \lfloor(\frac{k_2}{q}) \cdot ((\frac{q}{2}) \cdot \hat{\mathbf{M}}_b + \mathbf{v})\rceil\}.\end{aligned}$$

Therefore, $|\Pr[S_1]$ - $\Pr[S_2]| \leq \mathbf{Adv}_{n,q}^{RLWE}(\mathcal{B})$.

It follows that

$$\begin{aligned}\mathbf{Adv}_{PKE}^{IND-CPA}(\mathcal{A}) = |Pr[S_0] - Pr[S_2]| &\leq |Pr[S_0] - Pr[S_1]| + |Pr[S_1] - Pr[S_2]|\\ &\leq \mathbf{Adv}_{n,q}^{RLWE}(\mathcal{B}) + \mathbf{Adv}_{n,q,p}^{RLWR}(\mathcal{B}),\end{aligned}$$

which concludes the proof of Theorem 1.

Security of IND-CCA KEM We prove that our KEM scheme is IND-CCA secure under the public-key encryption in the quantum random oracle. Using Theorems 1 of Jiang et al. [23], we get the Theorem 2 for the IND-CCA security of our KEM in the quantum random oracle model.

Theorem 2 (IND-CCA KEM in QROM). *We define a public key encryption scheme PKE = (KeyGen, Encrypt, Decrypt) with message space $\mathcal{M}$ and which is (1-ϵ)-correct. For any IND-CCA quantum adversary $\mathcal{A}$ that makes at most q_D queries to the decryption oracle, at most q_G queries to the random oracle G and at most q_H queries to the random oracle H, we have that*

$$\boldsymbol{Adv}_{KEM}^{IND-CCA}(\mathcal{A}) \leq 2q_H \frac{1}{\mathcal{M}} + 4q_G\sqrt{1-\epsilon} + 2(q_G + q_H)\sqrt{\boldsymbol{Adv}_{PKE}^{IND-CPA}(\mathcal{B})}.$$

5.3 Cost of known attacks

TiGER128, TiGER192, and TiGER256 achieve the security levels 1(AES128), 3(AES192), and 5(AES256) suggested by the NIST [3], respectively, and the expected security strength for each parameter set is shown in Table 6.

The expected security strength is based on Core-SVP [6] and MATZOV [32]. It estimates using `LATTICE-ESTIMATOR` [5]. We also considered the Meet-LWE attack [33] particularly effective against sparse ternary secret.

We estimate the security strength of RLWR and RLWE, respectively, due to the character of the TiGER scheme, which uses a combination of RLWR for key generation and RLWE for encryption. The RLWR security strength estimate is equivalent to the LWE, whose noise distribution is uniform over the integers in the range $[-q/2p, q/2p)$ [11].

Estimating the security strength of ciphertext additionally considers deterministic noise added due to ciphertext compression in RLWE. For example, in TiGER128, if q is 2^8 and k_1 compressing ciphertext $\mathbf{c_1}$ is 2^7, a uniform distribution noise (e_{c1}) in the integer range of $[-1, 1)$ is added (stdev = 0.5) . If noise e_1 of RLWE generates $HWT(512, 32)$, the standard deviation is 0.25, so the standard deviation of the final noise is $e_1 + e_{c1} = 0.559$.

Table 6. Computational complexity of best known attacks

Schemes	TiGER128	TiGER192	TiGER256
Security level	1	3	5
Quantum Core-SVP	119	210	242
Classical Core-SVP	127	221	257
MATZOV	145	235	273
Meet-LWE	144 (mem.) 123	207 (mem.) 182	291 (mem.) 250

Core-SVP First, our estimates are based on the classical and the quantum Core-SVP hardness [6], which is a very conservative underestimation of the real security. Core-SVP is most commonly used in NIST competition, making it easy to compare algorithms. It also allows for a conservative approach in a quantum environment. The conservative estimation of classical Core-SVP is challenging to compare with the number of gates. For example, Kyber512 claims that the classical security strength 2^{118} based on Core-SVP can be converted to a classical gate of $2^{151.5}$ [8], and LightSaber also claims that the computational complexity of 2^{118} is converted into 2^{144} [17]. Due to this difficulty, there is still discussion in the pqc-forum about whether Kyber512, Kyber768, and Kyber1024 have the same security level as AES128, 192, and 256, respectively. [2]. Therefore, we set the parameters to have a higher Core-SVP estimate than Kyber, with a margin of 2^9 at security level 1 and 2^{38} at security level 3.

In a quantum environment, the impact of MAXDEPTH is a major factor in estimating security strength. NIST has described a plausible value range for MAXDEPTH from 2^{40} logical gates to 2^{64} logical gates [41]. Kyber claims that for the core-SVP-hardness operation estimates to match the quantum gate cost of breaking AES at the respective security levels, a quantum computer would need to support a maximum depth of 2^{70-80} [8]. The TiGER security strength estimates in quantum Core-SVP are sufficient considering the lower limit of the range of MAXDEPTH suggested by NIST. Also, TiGER has 2^{12}, 2^{44}, and 2^{10} higher than Kyber with the same security level, respectively. Remark, We report all security strength estimated by the `LATTICE-ESTIMATOR` [5].

MATZOV MATZOV [32] reported that classical gates are stricter than their estimates, Kyber512 has a security strength of $2^{137.5}$, and LightSaber has a security strength of $2^{138.4}$, which does not satisfy the security level. Although a script replicating MATZOV's results has not been released, Albrecht et al. implemented it in `LATTICE-ESTIMATOR` [5], estimating the security strength as 2^{140} for Kyber512 and $2^{137.8}$ for LightSaber, supporting MATZOV's result. After conducting a follow-up study, it was analyzed that MATZOV had been overestimated [16]. However, when considering TiGER use for national security purposes

[2] https://groups.google.com/a/list.nist.gov/g/pqc-forum

that prioritize security, it was a good idea to exceed NIST's required security level in the MATZOV estimate. As shown in Table 6, TiGER128, TiGER192, and TiGER256 exceed the classical security strengths of 2^{143}, 2^{207}, and 2^{272} suggested by NIST, respectively.

Meet-LWE We also specifically analyze Meet-LWE attacks. As our scheme is based on the RLWE(R) designed by sparse ternary distributions with Hamming weights, it is weak against the Meet-LWE attack [33]. The Meet-LWE attack improves on Odlyzko's Meet-in-the-Middle approach and targets the ternary secret used by TiGER. We estimate the security using the Meet-LWE python code by the Smaug team[3] and increasing the Hamming weight of the ternary secret to an appropriate.

5.4 Side channel attack

Timing attack It is well known that if a specific condition of a branch statement is linked to some secret information, an attacker to extract secret information such as a secret key or message. Implementations of TiGER do not use any secret-dependent branches or table lookups. It was also confirmed through Valgrind testing in the KPQClean project[4]. This means that typical implementations of TiGER are free from timing leakage. We have completed a constant-time implementation in computations that require secret information. This constant-time implementation is expected to also avoid Simple Power Attacks.

Other attacks For secure use of the scheme in the real world, side-channel attacks should also be considered based on stronger attacker assumptions such as Differential Power Analysis, Correlation Power Analysis, and fault attacks. When comparing TiGER and Kyber from a side-channel attacks perspective, the most significant differences are the method of sampling the secret key and multiplying polynomials. First, sparse secret sampling is used by promising homomorphic encryption schemes CKKS [13], and BIKE [7] a 4-round candidate for the NIST competition. It also shares Smaug, a KpqC candidate. Therefore, side-channel attacks and response techniques for sparse secret sampling are expected to continue to be studied, and Krausz et al.'s recent study [25] supports this expectation. Second, for the multiplication of special polynomials with ternary secrets, the Hiding method studied by Jung et al. [24] can be applied.

We believe that is not a specific problem to ours and will be resolved by the wisdom of crowds(as is the case of RSA and ECC).

6 Performance analysis

We evaluate performance (CPU cycles) using each reference code, and the evaluation environment is AMD Ryzen7 5700G @3.8GHz CPU, Ubuntu 22.04.1,

[3] https://kpqc.cryptolab.co.kr/smaug

[4] https://github.com/kpqc-cryptocraft/KPQClean

GCC 11.3.0 with option −O3, and the value is the average for 10,000 iterations. our implementation is available to https://github.com/honggoonin/TIGER.git. Meanwhile, for a fairer comparison, Smaug and our scheme, participating in KpqC, used the KPQClean project code.

6.1 Performance of reference implementation

TiGER is 2.34-2.88 times faster than the Kyber[5], 1.30-1.94 times faster than Saber[6], 1.22-1.82 times faster than Sable[7] and 1.26-1.71 times faster than the Smaug[8], with an equivalent security level. In particular, the higher the security strength, the larger the difference in performance between the TiGER and others. so, applied in national security, TiGER256 is best of them.

Table 7. Compare to Performance

Schemes	**CPU Cycles**				**Ratio**			
	Total	**KeyGen.**	**Encap.**	**Decap.**	**Total**	**KeyGen**	**Encap.**	**Decap.**
NIST Security Level 1 (AES128)								
TiGER128	**153,064**	**55,261**	**50,694**	**47,109**	**1**	**1**	**1**	**1**
Kyber512	357,805	91,427	118,863	147,515	2.34	1.65	2.34	3.13
LightSaber	199,728	61,731	67,944	70,053	1.30	1.12	1.34	1.49
LightSable	186,483	55,983	62,399	68,101	1.22	1.01	1.23	1.45
Smaug128	193,435	79,686	62,571	51,178	1.26	1.44	1.23	1.09
NIST Security Level 3 (AES192)								
TiGER192	**223,343**	**65,863**	**81,579**	**75,901**	**1**	**1**	**1**	**1**
Kyber768	559,623	154,013	183,957	221,653	2.51	2.34	2.25	2.92
Saber	350,817	104,453	120,082	126,282	1.57	1.59	1.47	1.66
Sable	325,984	95,862	110,136	119,986	1.46	1.46	1.35	1.58
Smaug192	289,515	114,809	90,688	84,018	1.30	1.74	1.11	1.11
NIST Security Level 5 (AES256)								
TiGER256	**280,954**	**72,634**	**105,362**	**102,958**	**1**	**1**	**1**	**1**
Kyber1024	810,271	234,497	264,973	310,801	2.88	3.23	2.51	3.02
FireSaber	545,817	164,286	186,304	195,227	1.94	2.26	1.77	1.90
FireSable	512,577	152,869	175,039	184,669	1.82	2.10	1.66	1.79
Smaug256	480,971	180,981	152,410	147,580	1.71	2.49	1.45	1.43

[5] https://github.com/pq-crystals/kyber.git
[6] https://github.com/KULeuven-COSIC/SABER
[7] https://github.com/josebmera/scabbard
[8] https://github.com/kpqc-cryptocraft/KPQClean

7 Conclude

The QR KEM design aims to replace cryptography in legacy security protocols. It would be nice if QR KEM had better performance and bandwidth than ECDH or DH for easy migration, but it seems impossible. Therefore, the maximum range of performance and bandwidth acceptable to the security protocol and network environment should be carefully considered and determined as the threshold of the scheme design. We decided the bandwidth threshold to be 1,442 bytes based on the IKEv2 over ethernet. Unfortunately, even schemes such as Kyber, standardized as FIPS 203, or Smaug as KpqC candidate have public key and ciphertext sizes exceeding 1,442 bytes at the AES256 security level used for national security.

In this paper, we propose TiGER, a Tiny bandwidth KEM for easy miGration based on RLWE(R). We design schemes with ciphertext and public key sizes smaller than 1,442 bytes each to simplify migration to the legacy security protocol. More precisely, TiGER256 with AES256 security level has 1,408 bytes of ciphertext and 928 bytes of public key. Additionally, by using a sparse ternary secret, the size of the secret key is significantly smaller, so it is expected to be well used in resource-constrained devices. Small ciphertext and public key size were achieved by choosing RLWR and RLWE as the base hardness, choosing a small modulus $q = 256$, and compress the ciphertext. TiGER is 2.34-2.88 times faster than Kyber and 1.18-1.64 times faster than Smaug. It is the advantage of choosing a sparse ternary secret uniform distribution. TiGER128, TiGER192, and TiGER256 satisfy security levels 1, 3, and 5, respectively, based on MATZOV, Core-SVP, and Meet-LWE attack. The decrypt failure probability of the TiGER design achieved a negligible by correcting errors using `XEf` and `D2` encoding.

Acknowledgments This work was submitted to the 'Korean Post-Quantum Cryptography Competition' (www.kpqc.or.kr).

References

1. Ajtai, M.: Generating hard instances of lattice problems. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. pp. 99–108 (1996)
2. Akleylek, S., Alkım, E., Tok, Z.Y.: Sparse polynomial multiplication for lattice-based cryptography with small complexity. The Journal of Supercomputing **72**(2), 438–450 (2016)
3. Alagic, G., Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Liu, Y.K., Miller, C., Moody, D., Peralta, R., et al.: Status report on the first round of the NIST post-quantum cryptography standardization process. US Department of Commerce, National Institute of Standards and Technology (2019)
4. Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Miller, C., Moody, D., Peralta, R., et al.: Status report on the third round of the nist post-quantum cryptography standardization process: Nistir 8413. NIST (2022)
5. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)

6. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange—a new hope. In: 25th {USENIX} Security Symposium ({USENIX} Security 16). pp. 327–343 (2016)
7. Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Gueron, S., Guneysu, T., Melchor, C.A., et al.: Bike. Round 2 submission to the NIST PQC Project (2019)
8. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber algorithm specifications and supporting documentation. NIST PQC Round **3**(4), 1–43 (2022)
9. Baan, H., Bhattacharya, S., Fluhrer, S.R., Garcia-Morchon, O., Laarhoven, T., Rietman, R., Saarinen, M.J.O., Tolhuizen, L., Zhang, Z.: Round5: Compact and fast post-quantum public-key encryption. IACR Cryptology ePrint Archive **2019**, 90 (2019)
10. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 719–737. Springer (2012)
11. Bogdanov, A., Guo, S., Masny, D., Richelson, S., Rosen, A.: On the hardness of learning with rounding over small modulus. In: Theory of Cryptography Conference. pp. 209–224. Springer (2016)
12. Cheon, J.H., Choe, H., Hong, D., Yi, M.: Smaug: Pushing lattice-based key encapsulation mechanisms to the limits. Cryptology ePrint Archive (2023)
13. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23. pp. 409–437. Springer (2017)
14. Cheon, J.H., Kim, D., Lee, J., Song, Y.: Lizard: Cut off the tail! practical post-quantum public-key encryption from lwe and lwr. Cryptology ePrint Archive, Report 2016/1126 (2016), https://eprint.iacr.org/2016/1126
15. D'Anvers, J.P., Tiepelt, M., Vercauteren, F., Verbauwhede, I.: Timing attacks on error correcting codes in post-quantum secure schemes. IACR Cryptology ePrint Archive **2019**, 292 (2019)
16. Ducas, L., Pulles, L.N.: Does the dual-sieve attack on learning with errors even work? In: Annual International Cryptology Conference. pp. 37–69. Springer (2023)
17. D'Anvers, J.P., Karmakar, A., Roy, S.S., Vercauteren, F.: Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem. In: International Conference on Cryptology in Africa. pp. 282–305. Springer (2018)
18. Fritzmann, T., Pöppelmann, T., Sepulveda, J.: Analysis of error-correcting codes for lattice-based key exchange. In: International Conference on Selected Areas in Cryptography. pp. 369–390. Springer (2018)
19. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Annual international cryptology conference. pp. 537–554. Springer (1999)
20. Gazdag, S.L., Grundner-Culemann, S., Guggemos, T., Heider, T., Loebenberger, D.: A formal analysis of ikev2's post-quantum extension. In: Annual Computer Security Applications Conference. pp. 91–105 (2021)
21. Hoffstein, J., Pipher, J., Silverman, J.H.: Ntru: A ring-based public key cryptosystem. In: International algorithmic number theory symposium. pp. 267–288. Springer (1998)

22. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the fujisaki-okamoto transformation. In: Theory of Cryptography Conference. pp. 341–371. Springer (2017)
23. Jiang, H., Zhang, Z., Chen, L., Wang, H., Ma, Z.: Ind-cca-secure key encapsulation mechanism in the quantum random oracle model, revisited. In: Annual International Cryptology Conference. pp. 96–125. Springer (2018)
24. Jung, C.G., Lee, J., Ju, Y., Kwon, Y.B., Kim, S.W., Paek, Y.: Lizarmong: Excellent key encapsulation mechanism based on rlwe and rlwr. In: International Conference on Information Security and Cryptology. pp. 208–224. Springer (2019)
25. Krausz, M., Land, G., Richter-Brockmann, J., Güneysu, T.: A holistic approach towards side-channel secure fixed-weight polynomial sampling. In: IACR International Conference on Public-Key Cryptography. pp. 94–124. Springer (2023)
26. Lee, E., Kim, Y.S., No, J.S., Song, M., Shin, D.J.: Modification of frodokem using gray and error-correcting codes. IEEE Access **7**, 179564–179574 (2019)
27. Lee, J., Kim, D., Lee, H., Lee, Y., Cheon, J.H.: Rlizard: Post-quantum key encapsulation mechanism for iot devices. IEEE Access **7**, 2080–2091 (2018)
28. Lee, J., Kim, S., Kim, C.H., Hong, S.: μ-hope: Compact size rlwe based kem using error correcting code. Journal of the Korea Institute of Information Security & Cryptology **30**(5), 781–793 (2020)
29. Lu, X., Liu, Y., Zhang, Z., Jia, D., Xue, H., He, J., Li, B., Wang, K., Liu, Z., Yang, H.: Lac: Practical ring-lwe based public-key encryption with byte-level modulus. IACR Cryptology ePrint Archive **2018**, 1009 (2018)
30. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 1–23. Springer (2010)
31. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-lwe cryptography. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 35–54. Springer (2013)
32. MATZOV: Report on the Security of LWE: Improved Dual Lattice Attack (Apr 2022). https://doi.org/10.5281/zenodo.6493704, https://doi.org/10.5281/zenodo.6493704
33. May, A.: How to meet ternary lwe keys. Cryptology ePrint Archive, Paper 2021/216 (2021), https://eprint.iacr.org/2021/216, https://eprint.iacr.org/2021/216
34. Mera, J.M.B., Karmakar, A., Kundu, S., Verbauwhede, I.: Scabbard: a suite of efficient learning with rounding key-encapsulation mechanisms. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 474–509 (2021)
35. Park, S.H., Kim, S., Lee, D.H., Park, J.H.: Improved ring lwr-based key encapsulation mechanism using cyclotomic trinomials. IEEE Access **8**, 112585–112597 (2020)
36. Peikert, C., Regev, O., Stephens-Davidowitz, N.: Pseudorandomness of ring-lwe for any ring and modulus. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing. pp. 461–473. ACM (2017)
37. Peikert, C., Regev, O., Stephens-Davidowitz, N.: Pseudorandomness of ring-lwe for any ring and modulus. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing. pp. 461–473 (2017)
38. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
39. Saarinen, M.J.O.: Hila5: On reliability, reconciliation, and error correction for ring-lwe encryption. Cryptology ePrint Archive, Report 2017/424 (2017), https://eprint.iacr.org/2017/424

40. Smyslov, V.: Intermediate exchange in the ikev2 protocol. IETF draft (2021)
41. of Standards, N.I., Technology: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016)
42. Targhi, E.E., Unruh, D.: Post-quantum security of the fujisaki-okamoto and oaep transforms. In: Theory of Cryptography Conference. pp. 192–216. Springer (2016)
43. Walters, M., Roy, S.S.: Constant-time bch error-correcting code. IACR Cryptology ePrint Archive **2019**, 155 (2019)
44. Zimmer, D.I.E.: Post-quantum kryptographie für ipsec. In: Sicherheit in vernetzten Systemen-22. DFN-Konferenz. Ed. by Christian Paulsen. DFN-CERT (2015)